WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

A Group Theoretic Integer Programming Algorithm: System Design and Computational Experience

by

G. Anthony Gorry, William D. Northup
Jeremy F. Shapiro, Clark H. Zakovi

513-71
February 1971

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

A Group Theoretic Integer Programming Algorithm: System Design and Computational Experience

by

G. Anthony Gorry, William D. Northup
Jeremy F. Shapiro, Clark H. Zakovi

513-71
February 1971

Introduction

In this paper we report on a new integer programming algorithm (IPA). The central idea upon which this algorithm is based is that the integer solution to a system of linear equations can be usefully characterized by transforming the system to an equation of elements from a finite abelian group. In order to apply this classical approach of number theory to integer programming (IP) problems, it is necessary to take into account non-negativity constraints and the selection of an optimal solution according to some objective function. The details of this theory upon which IPA is based are discussed in Gorry and Shapiro (1). In this paper we will be less concerned with theoretical details and more concerned with the practical aspects of implementing the algorithm as an integer programming system. In addition we will give some computational results which are indicative of our experience with a variety of real world integer programming problems. Computational experience has indicated that group theoretic IP methods provide qualitative as well as quantitative insights into the structure of integer programming problems.

Our plan for this paper is as follows. First, we will give a general overview of the way in which IPA operates. In that discussion we will briefly describe each of the major subroutines of the system. Basically, these are the subroutines that are required to: (1) obtain an optimal solution to the continuous version of the integer programming problem, (2) transform the problem to a group optimization problem, (3) solve one of three possible group optimization problems and (4) use the resulting solution to the group problem in a search for an optimal solution to the IP problem. As noted, each of these functions is supported by one or more

subroutines in the system. For each the general purpose and means of operation will be discussed, and in addition, we will give some representative computational experience. Then we will discuss the way in which the system can be used to manipulate the data of integer programming problems. This data manipulation allows the system to find an optimal solution to the given IP problem with less computational effort. Finally, we discuss some of our experience in solving large integer programming problems.

The system we will be discussing is currently implemented in FORTRAN IV. We chose to implement the system in FORTRAN because we desired to have the ability to transfer the system from one computer to another. We have found this ability particularly useful, and to date the system has been implemented on various versions of the IBM 360 System (Models 65, 67, 75 and 85) and on the Univac 1108. Currently, we are in the process of implementing this system on the CDC 6600 machine. We have made the more obvious attempts to optimize the encoding of this system in that we have been concerned with data structures and sequencing of operations within the system. On the other hand, we have not implemented any of this system in assembly language for a given machine. We do feel that such an implementation would significantly improve the computational efficiency of our system but for our purposes the transferability of the FORTRAN version has definite appeal.

Overview of IPA

We begin with a statement of the IP problem in its initial form. This problem is

$$\min z = \sum_{j=1}^{n+m} c_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n+m} a_{ij} x_j = b_i \quad i = 1,\ldots,m \tag{1}$$

$$x_j = 0,1,2,\ldots \quad j \in U$$

$$x_j = 0 \text{ or } 1 \quad j \in Z$$

where the coefficients c_j, a_{ij}, b_i are integers, U and Z are disjoint subsets of $\{1,2,\ldots,n+m\}$, and $U \vee Z = \{1,2,\ldots,n+m\}$. Let a_j denote a generic m vector with components a_{ij}, and let b be the m vector with components b_i. Problem (1) is solved first as a linear programming (LP) problem by a simplex algorithm. Without loss of generality, assume the optimal LP basic variables are x_{n+i}, $i = 1,\ldots,m$, and the non-basic variables are x_j, $j = 1,\ldots,n$. For convenience, define $y_i \equiv x_{n+i}$, $i = 1,\ldots,m$. Thus, the optimal LP basis B given by

$$B = \begin{pmatrix} a_{1,n+1} & \cdots & a_{i,n+m} \\ & & \\ a_{m,m+1} & \cdots & a_{m,n+m} \end{pmatrix} \tag{2}$$

is used to transform (1) to the equivalent form

$$\min z = z_0 + \sum_{j=1}^{n} \bar{c}_j x_j \tag{3a}$$

$$\text{s.t. } y_i = \bar{b}_i - \sum_{j=1}^{n} a_{ij} x_j \quad i = i,\ldots,m \tag{3b}$$

$$y_i = 0,1,2..., \quad n + i\varepsilon U \tag{3c}$$

$$y_i = 0 \text{ or } 1 \quad n + i\varepsilon Z \tag{3d}$$

$$x_j = 0,1,2... \quad j\varepsilon U \tag{3e}$$

$$x_j = 0 \text{ or } 1 \quad j\varepsilon Z \tag{3f}$$

where $\bar{c}_j \geq 0$, $j = 1,...,n$, $\bar{b}_i \geq 0$, $i = 1,...,m$, $\bar{b}_i \leq 1$, $n + i\varepsilon Z$.

The optimal LP solution is $x_j^* = 0$, $j = 1,...,n$, $y_i^* = \bar{b}_i$, $i = 1,...,m$. Clearly, the original IP problem (1) is solved if y_i^* is integer for all i. If one or more basic variables are not integer in the optimal LP solution, then some of the independent non-basic variables x_j must be set at positive integer values in order to make the dependent basic variables integer and non-negative. We call a non-negative integer vector $x = (x_1,...,x_n)$ satisfying (3e) and (3f) a correction to the optimal LP solution. If the resulting y_i given by (3b) satisfy (3c) and (3d), then x is a feasible correction. The least cost feasible correction is an optimal or minimal correction.

Group theory is used to transform problem (3) to another equivalent form. Some details of this transformation are given below, and at this point we simply state the result. Each non-basic activity a_j is mapped into an r-tuple of integers $\alpha_j = (\alpha_{1j},\alpha_{2j},...,\alpha_{rj})$ where $0 \leq \alpha_{ij} < q_i$, $i = 1,...,r$, and q_i is an integer greater than or equal to 2. Similarly, the right hand side vector b is mapped into the r-tuple of integers $\beta = (\beta_1,...,\beta_r)$ where $0 \leq \beta_i < q_i$, $i = 1,...,r$.

The equivalent formulation is

$$\min z = z_0 + \sum_{j=1}^{n} \bar{c}_j x_j \tag{4a}$$

$$\text{s.t. } y_i = \bar{b}_i - \sum_{j=1}^{n} \bar{a}_{ij} x_j \qquad i = 1,\ldots,m \tag{4b}$$

$$\sum \alpha_j x_j \equiv \beta \bmod(q_1,\ldots,q_r) \tag{4c}$$

$$y_i \geq 0, \quad n + i \in U \tag{4d}$$

$$y_i \leq 1, \quad n + i \in Z \tag{4e}$$

$$x_j = 0,1,2\ldots \quad j \in U \tag{4f}$$

$$x_j = 0 \text{ or } 1 \quad j \in Z \tag{4g}$$

Problem (4) is the starting point for the group theoretic approach. It is difficult to explicitly consider all the constraints in (4), and IPA progresses by ignoring certain constraints and solving the related problems which result. The optimal solutions to these problems are then used in a manner to be described later to find an optimal solution to (4). We simply mention here that this method of ignoring some constraints thereby creating easier optimization problems is called relaxation. Specifically, if X is the set of feasible corrections, then the algorithm will usually deal with sets $X < X'$. Then, if a correction x^* which is optimal in $\sum_{j=1}^{n} \bar{c}_j x_j$ subject to $x \in X'$ is feasible in (4), (i.e., $x^* \in X$), then x^* is optimal in (4). Moreover, $\sum_{j=1}^{n} \bar{c}_j x_j^* \leq \sum_{j=1}^{n} \bar{c}_j x_j$ for all feasible corrections $x \in X$. These properties of relaxation are the basis for the branch and bound search of IPA, and also for the data manipulation schemes.

IPA Linear Programming

Within the IPA system there are two ways to solve the integer programming problem as a continuous problem. First, as a subroutine within IPA there is an upper bounding variant of the simplex algorithm. This routine is written in FORTRAN IV and can be used effectively when the problem in question has relatively few rows. It is not easy to make this number of rows precise because to a certain extent it depends on the character of the problem, but basically we can say that for problems of fewer than 200 to 250 rows, our experience has been that this routine is satisfactory. One problem which can arise, however, is a numerical one. The IPA linear programming algorithm employs single precision floating point arithmetic. As a result, round-off problems can be encountered if the determinants of the bases for the continuous problem are large. For reasons which will be explained below, however, this problem is not as severe as it might seem. In general, we attempt to use data manipulation methods to restrict the size of determinants because of their effect on the size of the induced group. Because these methods are generally effective, we have found this numerical problem to be less important than might be expected.

A second method which is available for the solution of the continuous problem is the use of any LP system which will obtain an optimal continuous solution. The use of a variety of LP algorithms is possible because the IPA system is written in such a way as to accept a specification (in a certain format) of the optimal continuous solution regardless of how that solution was obtained. Currently, the only restriction on the method used to solve the continuous problem is that the resulting specification of the optimal solution conform to that employed with the IBM MPS linear programming system.

This format was chosen because of the widespread availability of this system. Parenthetically, we should note that we have developed an interface program which permits us to take solutions derived by MPS directly into IPA as the first step in solving an integer programming problem.

In general, we would prefer the use of such a system as MPS for relatively large problems because we have not devoted any particular attention to the detailed procedures used by our LP algorithm. This is because our emphasis has been on the solution of the residual problem in order to find an integer solution. In the case where the user of IPA has available to him a relatively efficient LP code, he would be well advised to use it in conjunction with IPA because in most cases he would gain computational efficiency. On the other hand, our linear programming code does have certain advantages in permitting certain data manipulation strategies for integer programming problems as we will see below.

Regardless of the approach which is used to solve the LP problem, the IPA requirements are the same. Specifically, the system requires a specification of the original problem and a list of the basic variables which were in the optimal tableau for the LP problem. These data are sufficient to permit IPA to begin the search for an optimal integer solution to the problem.

The Group Representational Algorithm

The group representational algorithm (GETREP) is based on some classical ideas about matrix diagonalization. The transformation of a given IP problem to a group problem occurs after the optimal LP solution has been obtained and found to be non-integer. The transformation is effected by ignoring the non-negativity and upper bound constraints on the basic variables and characterizing the set of all corrections x which make the basic variables integer.

We can restate in the terminology of number theory the requirement that the y_i be integer in (3b) by requiring that the x_j satisfy the system of congruences

$$\sum_{j=1}^{n} \bar{a}_{ij} x_j \equiv \bar{b}_i \pmod 1, \quad i = 1,\ldots,m$$

$$x_j = 0, 1, 2, \ldots \quad , \quad j \in U$$

$$x_j = 0 \text{ or } 1, \quad j \in Z \qquad (5)$$

The columns of transformed non-basic activities $\bar{a}_j = (\bar{a}_{ij})$ generate a finite abelian group under addition (modulo 1). It is easy to see that the integer parts $[\bar{a}_{ij}]$ of the coefficients a_{ij} can be ignored when trying to satisfy the system of congruences. From this point of view, the set of elements in the group contains all the possible fractions which can be generated by taking integer combinations of the columns $\bar{a}_j$. For each row i, the fraction generated on the left in equation ( ) must equal the fraction $\bar{b}_i - [\bar{b}_i]$ on the right. The algorithm actually computes the group G generated by the columns of B^{-1} with addition (modulo 1). G has $D = |\det B|$ elements. The previous group mentioned

above is a subgroup of G, and it is almost always G itself.

The efficiency of the group theoretic approach depends to a large extent on the fact that the system of m congruences in (5) can be greatly reduced in number. In particular,

$$G \cong Z_{q_1} \oplus Z_{q_2} \oplus \ldots \oplus Z_{q_r} \tag{6}$$

where

q_i integer, $q_i \geq 2$ $\quad i = 1,\ldots,r$

$q_i | q_{i+1}$ $\quad i = i,\ldots,r - 1$

$$\prod_{i=1}^{r} q_i = D = |\det B|$$

Z_{q_i} = residue class of the integers modulo q_i.

The number of terms r in the direct sum decomposition is always less than or equal to m, and usually r = 1, 2 or 3. Moreover, r is the minimal number of terms which can be used in an isomorphic representation of G. See below for representative group structures.

The cannonical representation of G is achieved by a reduction of the m x m matrix $F = f_{ki} = D\{\bar{a}_{k,n+i} - [\bar{a}_{k,n+i}]\}$, k = 1,...,m, i = 1,...,n. Through a series of passes through F, the matrix is diagonalized. Each of these passes consists of finding the minimum non-zero element in F and using the row and column in which that element exists for elementary row and column operations on F. These row (column) operations are like the typical Gauss-Jordan reduction steps except that the constant multiple of a row (column) which is to be subtracted from another row (column) is determined in integer arithmetic and the reductions are made

modulo D. In the event that a pass through the matrix leaves a given row and column with only one non-zero element at their intersection, the row and column are in general removed from any further consideration in the diagonalization procedure. The only exception to this is when the non-zero intersection element does not divide the determinant or does not divide the similar preceding element. In such a case the representational algorithm "backs up" and continues the process. We will omit here details of this last maneuver. Basically, the purpose for it is to insure that the minimal group representation is obtained by the algorithm. The diagonalization procedure is complete when the product of the non-zero intersection element equals the determinant of the basis matrix.

This procedure can be shown to be equivalent to the following matrix operation. A matrix of row operations, R, and a matrix of column operations, C, can be used to diagonalize the matrix F. If we denote by δ the resulting diagonal matrix we can say then:

$$\delta = RFC$$

After we have completed the diagonalization of F, the group identities of the non-basic vectors for the integer programming problem can be obtained by the following relationship:

$$\alpha_j = C^{-1} a_j$$

Notice that if the diagonalization procedure has revealed r subgroups (that is, if r elements were required to form a product equal to the determinant), then the α's are determined by the first r innerproducts of rows of C^{-1} and the non-basic columns taken modulo the corresponding subgroup orders.

We have experimented with various procedural realizations of the representational algorithm. The basic difficulty here is a classic one in computer programming. To the extent that one uses large blocks of core storage to hold the various matrices involved, once can improve the speed with which the algorithm operates. On the other hand, for problems of say, 200 rows or more, the core storage requirements for this approach can be very great and as a result, the system may be forced to use secondary storage. In such a case there will be a decline in the overall speed of the representational algorithm. Therefore, we have been concerned with developing procedures which will permit us to minimize the amount of storage required for the algorithm and at the same time maintain reasonable computational efficiency. The current version of this algorithm which is used in the IPA system represents a compromise between speed and core storage requirements. Our experience with this version of the algorithm has been most satisfactory.

In Figure 1 we have presented some representative times for the algorithm on a variety of integer programming problems. As can be seen from this figure, in spite of the fact that the algorithm is written in FORTRAN, the computational requirements to obtain group representation are relatively small. We have not performed any detailed study on the relationship between the computational time required for the representational algorithm and such variables as the number of rows in the problem or the number of non-basic columns. There are, however, certain general observations we can make based on the data presented. In general the computational requirement increases with the number of rows in the integer

Figure 1. Group Representational Algorithm: Computational Experience

(*denotes IBM 360/67; all others Univac 1108)

	Number of Rows m	Determinant D	Number of Subgroups r	Time in Seconds t	Number of Non Basics n
1	6	180	2	0.90	240
2	6	1080	3	0.92	240
3	6	10	1	1.13	300
*4	9	128	3	0.06	14
5	12	5832	6	0.35	104
6	12	14400	5	0.38	104
7	18	81600	3	1.14	240
8	18	55296	5	1.15	240
9	18	900	2	1.11	240
10	18	1158	1	1.01	240
11	18	960	2	1.04	240
12	18	23328	4	1.14	240
13	18	96	2	1.05	239
*14	36	384	4	0.91	36
*15	36	144	2	0.73	36
*16	46	420	1	3.41	115
*17	50	21744	1	2.19	115
*18	50	36288	1	2.90	115
19	86	280	2	1.46	109
20	86	140	1	1.40	109
21	86	1080	1	1.82	109
22	86	17100	2	1.53	109
23	86	540	2	1.37	109
24	86	380	1	1.60	109
25	91	2048	3	1.67	104
26	91	512	3	1.45	104

programming problem because the number of elements in the matrix F which must be considered is equal to the square of the number of rows. Thus, increasing the number of rows sharply increases the amount of computation required to diagonalize the fractional matrix. On the other hand, our experience to date has not shown the number of subgroups to be an important factor in determining the computational time. The reason for this is that a single pass through the fractional matrix will produce many zero elements in that matrix. Subsequent passes through the matrix will then generally find a number of constant multiples for rows and columns which are zero and hence effectively can be ignored. The number of non-basic columns in the problem is also an important determinant of the computational time. Once C^{-1} has been obtained, the calculation of the group identities for the non-basics requires the multiplication of some number of rows of C^{-1} by each of the non-basic columns in turn. As a result, if the number of non-basics is quite large, then the computational time for the representational algorithm will also be relatively large. Notice, however, that these times will show only a relatively large increase. In other words, the absolute time required to obtain the representation is in general quite small and for the problems which we have dealt with where the number of rows and columns is less than say 500, this time is on the order of one to three seconds.

Group Optimization Problems

After GETREP has obtained the canonical group representation IPA solves one of three group optimization problems. These problems are constructed by ignoring some of the constraints in problem (4), and therefore they are relaxations of (4). As mentioned in the overview, this approach will provide either an optimal correction in (4), or the necessary information about lower bounds and trial correctio s from which to do an efficient search for an optimal correction.

The first group optimization problem is the unconstrained group problem: Find G_u where

$$G_u = \min \sum_{j=1}^{n} \bar{c}_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} \alpha_j x_j \equiv \beta \bmod(q_1,\ldots,q_r) \qquad (7)$$

$$x_j = 0, 1, 2,\ldots; \quad j = 1,\ldots,n$$

Problem (7) can be solved by a dynamic programming algorithm based on the recursion (see reference (4))

$$F(\lambda_k) = \min_{\substack{j = 1,\ldots,n \\ k = 1,\ldots,n}} \{\bar{c}_j + F(\lambda_k - \alpha_j)\} \qquad (8)$$

$$F(\lambda_0) \equiv 0$$

The dynamic programming calculations and the resulting optimal solutions are best described in network terminology. The network has D nodes, one for each $\lambda_k \varepsilon G$. There are directed arcs $(\lambda_k - \alpha_j, \lambda_k)$, $j = 1,\ldots,n$,

drawn to every node λ_k except the node $\lambda_0 \equiv 0$. The arc length or cost associated with an arc of type j is $\bar{c}_j$. Problem (7) in this setting is the problem of finding a shortest route from λ_0 to β. In the process of finding this particular shortest route path, the algorithm find the shortest route from λ_0 to all λ_k, k = 1,...,D - 1. As we will discuss below, these additional paths are important if a search is required. An optimal solution w to (7) is extracted from the shortest route path by letting w_j equal the number of times an arc of type j is used in the optimal path. If w is a feasible correction, it is an optimal correction. Some computational experience with the unconstrained group problem is given in Figure 2.

Clearly, w may fail to be feasible because some of the constraints (4d), (4e) or (4g) may be violated. Of these constraints, the 0-1 constraints (4g) on the non-basic variables can be explicitly considered when solving the shortest route group problem. This gives us the second of the group problems, called the zero-one group optimization problem. This algorithm is a generalization of the shortest route network algorithm for the unconstrained group problem. The basic difference is that the 0-1 non-basics may be used at most once in extending low cost paths in the network.

Because it accounts for 0-1 constraints on the non-basic variables, the zero-one group algorithm gives better bounds for the branch and bound search. On the other hand, more computation and storage is required for the solution of this problem. In general, more than one path through a given node in the network must be maintained. This is because the shortest route path from λ_k to 0 is not necessarily part of the shortest route

Figure 2 Solution of the Unconstrained Group Optimization Problem: Computational Experience

(*denotes IBM 360/67; all others Univac 1108)

	Number of Non Basics n	Determinant D	Number of Subgroups r	Time in Seconds t		Number of Non Basics n	Determinant D	Number of Subgroups r	Time in Seconds t
1	300	10	1	0.06	20	109	280	2	1.93
2	104	12	1	0.04	21	104	288	4	1.00
3	104	24	2	0.07	22	109	380	1	1.18
4	104	24	2	0.18	23	109	380	1	1.40
5	240	24	2	0.20	*24	36	384	4	1.16
6	104	44	1	0.19	25	104	512	3	5.86
7	104	48	3	0.14	26	109	540	2	2.53
8	239	96	2	0.96	27	240	900	2	12.34
9	240	120	3	1.11	28	240	960	2	3.83
*10	36	144	2	0.30	29	104	972	3	1.83
11	109	140	1	0.58	30	109	1080	1	4.20
*12	14	128	3	0.27	31	240	1080	3	7.89
13	104	144	2	0.76	32	240	1158	1	3.98
14	109	168	2	0.89	33	109	1260	2	12.50
15	240	180	2	0.68	34	240	1440	3	8.21
16	104	192	2	0.64	35	104	2048	3	12.88
17	104	192	2	0.65	36	103	3780	2	7.62
18	109	200	2	0.95	37	104	4752	2	13.27
19	240	240	3	1.22					

path from x_j to 0 when the latter passes through λ_k. For example, the price of this path from λ_j to λ_k and the shortest route path from λ_k to 0 may both use an arc corresponding to a 0-1 variable. Hence the algorithm must develop many more paths in general than the number required in the unconstrained case. Similarly the storage requirements are greater. To date our experience indicates that this algorithm requires about 3 to 5 times as much computation and about 5 times as much storage. Therefore, we use it only when the 0-1 constraints on non-basics are important (e.g. these variables have low $\bar{c}_j$ and tend to be heavily used in the network).

The final group problem is called the dynamic unconstrained group problem. A series of problems similar to (7) result as certain variables in problem (7) are fixed at zero. Problems such as these arise during the search discussed below. Since the set of variables being fixed at zero can be considered as increasing, the algorithm can compute a new optimal tree from the one previously computed.

Search Procedures in IPA

Above we have discussed in general the construction and solution of the group problems as relaxations of the original integer programming problem. Only one group problem is used during a given search. This relaxed problem is used extensively by the IPA search routine. In this section we will discuss the theoretical notions underlying the use of this problem, and we will draw heavily on the detailed considerations in Gorry and Shapiro (1). Then we will outline some of the computational procedures employed to realize this theory.

Several of our definitions above will be useful here. Recall that we defined a correction as any non-negative integer setting of the non-basic variables such that the upper bound constraints for these variables are satisfied. Any correction x such that y_i from (3b) satisfy (3c) and (3d) is called a feasible correction. A minimal cost feasible correction is an optimal correction.

For the majority of interesting integer programming problems, it is necessary to enumerate or search the set of corrections in a nonredundant and implicitly exhaustive fashion. To generate the nonredundant tree of corrections, we proceed as follows. Starting at K = 0, the corrections at level K are the corrections x such that

$$\sum_{j=1}^{n} x_j = K$$

Let x' be an arbitrary correction at level K and let j'(x), j(x) be defined by:

$$j(x') = j'(x') - 1 \quad \text{if } j'(x') \varepsilon Z$$

$$= j'(x') \qquad \text{if } j'(x') \varepsilon U$$

where

$$j'(x') = \min\{j \mid x'_j > 0\}$$

The correction x'_j is corrected to level K + 1 by continuing x' to $x' + e_j$ for $j \leq j(x')$ where e_j is the jth unit vector in n-space. As it is described, the tree of enumerative corrections is infinite. If upper bounds on the x_j are known, however, then the tree can be made finite by not permitting any corrections to be generated for which an upper bound is violated.

In general, the search procedure works as follows. If a given correction x' yields integral values for the y_i through the equations (3b), then it is only necessary to test the resulting y_i for non-negativity. A necessary condition for integrality is that

$$\bar{b}_i - \sum_{j=1}^{n} \alpha_j x'_j \equiv 0 (\text{mod } q_1, \ldots, q_r)$$

If the y_i corresponding to x' are non-integral, then the problem is to find a continuation vector w such that x' + w makes the y_i integer (and non-negative). The vector w must be such that the "incremental cost" of obtaining integrality for y_i ($\bar{c}w$) is minimized. Let

$$\bar{b}_i(x') = \bar{b}_i - \sum_{j=1}^{n} \bar{a}_{ij} x'_j, \quad i = 1, \ldots, m$$

and

$$\beta(x') = \beta - \sum_{j=1}^{n} \alpha_j x'_j$$

then this problem is

$$\min \sum_{j=1}^{n} \bar{c}_j w_j$$

p.t. $$y_i = \bar{b}_i(x') - \sum_{j=1}^{n} \bar{a}_{ij} w_j, \quad i = 1,\ldots,m$$

$$\sum_{j=1}^{n} \alpha_j w_j = \beta(x')$$

$$w_j = 0, \quad j = j(x') + 1,\ldots,n$$

$$w_j = 0 \text{ or } 1, \quad j \in Z \cap \{1,\ldots,j(x')\}$$

$$w_j = 0,1,2,\ldots, \quad j \in U \cap \{1,\ldots,j(x')\}$$

$$y_i \geq 0, \quad i = 1,\ldots,m$$

$$y_i \leq 1, \quad n + i \in Z$$

During an implicitly exhaustive search of the tree of enumerated corrections, the least cost correction x* found to date is called the incumbent. The incumbent cost is $z^* = \bar{c}x^*$. We say that an enumerated correction, x', has been fathomed if it is possible either:

(1) to discover that there exists no feasible correction of the form x' + w, w non-negative integer such that $\bar{c}(x' + w) < z^*$,

(2) to find a feasible correction of the form x' + w, w non-negative integer, such that $\bar{c}(x' + w) < z^*$. In this case, $x^* \leftarrow x' + w$ and $z^* \leftarrow \bar{c}(x' + w)$.

In either case, if x' is fathomed, then the entire subtree beneath x' has been implicitly tested, and x' is not continued.

The fathoming tests used by the IPA search routine are based on the solution of one of the group problems.

The first problem which can be used to try to fathom x' is the unconstrained group problem. Let v' be any correction such that $v' \leq x'$. An important special case is v' = 0. The unconstrained group problem is: Find $G_u(x',v')$ where

$$G_u(x',v') = \min \sum_{j=1}^{n} \bar{c}_j w_j$$

$$\text{s.t.} \quad \sum_{j=1}^{n} \alpha_j w_j \equiv \beta(x') \bmod(q_1,\ldots,q_r)$$

$$w_j = 0 \quad j = j(v') + 1,\ldots,n$$

$$w_j = 0,1,2,\ldots, \quad j = 1,\ldots,j(v')$$

and

$$\beta(x') = \beta - \sum_{j=1}^{n} \alpha_j x'_j$$

Let $w_u(x',v')$ be the optimal solution to this problem, then

Lemma The correction is fathomed if either

(1) $\bar{c}x' + G_u(x',v') \geq z^*$ or

(2) $\bar{c}x' + G_u(x',v') < z^*$ and $x' + w_u(x',v')$ is a feasible correction.

In the latter case, $x' + w_u(x',v')$ becomes the new incumbent. The use of the zero-one group problem is exactly the same except that the group problem is stated with bounds on the number of times certain arcs can be used in a given path in the network. Also the same is the use of the dynamic version of the unconstrained group problem, except that here certain variables are prohibited from entering into network paths by

constraining them to be zero.

The search procedure exploits this theory in the following way. An attempt is made to fathom the initial correction $x' = 0$ through the use of one of the group problems. In the event that a feasible correction is found in this way, the problem is solved. Otherwise, the correction $x' = 0$ must be continued. The resulting corrections are placed on a "correction list" and the parent correction ($x' = 0$) is deleted. Then a new correction is selected (currently the first one found on the list) and the process is repeated. Thus in the absence of an incumbent, no correction is removed from the list until its continuations have been generated and saved.

When a correction is saved, its bound ($\bar{c}x' + G.(x',v')$) is saved with it. Whenever this correction is retrieved from the list, this bound is tested against the incumbent value if one exists. If the correction is fathomed by bound, it is deleted directly. Thus, although a large number of corrections may be generated during the search, the number of corrections backtracked may be very much less if an incumbent has been found.

Through the use of secondary storage, the system has the capacity to manage an indefinitely large list of corrections. On the other hand, because of the potentially large number of corrections to be considered, the efficiency of the system with respect to its consideration of a single correction is important. Therefore, we have made an attempt to minimize the number of corrections which cannot be fathomed directly and the time required to conduct fathoming tests when necessary.

One way in which the number of corrections generated can be reduced is through the use of backtracking information. For example, given a

backtracked completion, w, of a correction x' from the unconstrained group problem solution, we can eliminate any continuation x" if $x" \leq x' + w$ element by element. This is because any such w represents a node which is contained in optimal path from $\beta(x')$ to 0. If we were to list this correction, we would only be "unwinding" the path from $\beta(x')$ to 0, and we would not be discovering any new potentially feasible corrections. (See (2) for a detailed discussion of this strategy.) In general, we have found that this "pruning" of the tree of corrections reduces the total number of corrections considered during the search by 10 - 15%. This strategy of "jumping over" corrections has a second major value. It permits the system to get deeper in the tree of corrections in a given period of time. This increases the probability that an incumbent will be found for a given amount of computation. Finding an incumbent sooner can cause a dramatic reduction in the number of corrections generated.

In order to improve the efficiency of the system in dealing with individual corrections when backtracking is required, the system encodes solutions to the group optimization problems into a threaded list structure. This permits the rapid retrieval of the "real-space" equivalent of backtracked paths from the network.

The question of when a given correction should be backtracked is important. Basically, the options are to backtrack a correction before it is stored on the correction list or after it is retrieved from that list. The advantage of the first strategy is that if an incumbent is found which can fathom the correction in question by bound, it need

never be backtracked. Hence, large numbers of corrections hopefully can be placed on the list without backtracking and later be fathomed by bound. In this way, the search time can be decreased significantly.

On the other hand, by failing to backtrack a given correction before it is placed on the list, we postpone the opportunity to discover whether it is a new incumbent. At the time the correction is generated (as a continuation of a parent correction), many thousands of corrections may be in the list. In its current mode of operation, the system must consider all these corrections before the given correction is retrieved from the list. In the case that the given correction is indeed a new incumbent, this is obviously undesirable.

Because of the importance of finding an incumbent early in the search, the IPA system backtracks corrections as they are generated (before they are placed on the correction list). Our experiments with this rule have shown it to be markedly superior to the alternative of deferring backtracking.

Again in the search, we have faced the tradeoff between core storage requirements and execution time. For each correction, in general, we need: 1) x', 2) $G.(x',v')$, 3) $\beta(x')$ and 4) $\bar{b}(x')$. If we were to store the last quantity with each correction we would rapidly be forced to secondary storage. To see this, consider how rapidly core storage would be exhausted if the average correction had 500 continuations, and $\bar{b}(x')$ had 300 rows. Because of this problem, IPA generates $\bar{b}(x')$ as required for each x' from x' and the vectors $B^{-1}a_j$. Of course, if a correction is fathomed by bound, no such generation is required. The other quantities, x', $G.(x',v')$ and $\beta(x')$ are stored with each correction.

In Figure 3 we have assembled some representative search times for IPA. From this Figure, the basic relation of the search time to the number of corrections can be seen. The number of rows in the problem is also important, however, because of the way in which feasibility tests are conducted. If a correction is not fathomed by bound, then the backtracked completion, w, is obtained and added to the correction. The resulting vector $x = x' + w$ is used to generate y_i's. As soon as one of the y_i is infeasible (either because it is negative or because it exceeds its upper bound), the solution is found to be infeasible. If the number of infeasibilities is a small fraction of the number of rows, then the expected number of y_i which must be computed to find an infeasibility is relatively large. If this situation occurs very frequently during the search, then the overall computational time will be relatively large.

One approach to this problem is to reorder the rows of the problem to increase the probability that any infeasibility will be found in the first few rows which are checked. The system currently does not do such reordering, but we are planning to introduce some on an experimental basis. We plan to use the infeasibilities from the initial backtracked solution ($x' = 0$) to isolate potential feasibility problems. The reason that this approach seems promising is that often a given "part" of the problem will yield feasibility problems for large numbers of corrections. A better strategy undoubtedly would be to dynamically reorder the rows of the problem as the search proceeds in accordance with observed patterns of infeasibilities. We hope to develop such a strategy in the future, but as yet our thoughts on this matter are only preliminary.

Figure 3. Search Times

	Rows	Non-basics	Number of Corrections	Time (sec)	Machine
1	12	104	99	0.77	1108
2	9	12	6769	7.14	360/85
3	36	36	7979	103.20	360/67
4	9	14	8463	19.63	360/67
5	12	104	11570	67.68	1108
6	9	14	12814	7.51	360/85
7	14	13	20692	10.51	360/85
8	12	104	169902	253.00	1108
9	313	170	668420	361.85	360/85

There is one other aspect of the search which is worth mentioning here. The system permits the introduction of a "pseudo-incumbent" prior to the search. This can be either a known feasible integer solution or a guess at the cost of such a solution. If the pseudo-incumbent is reasonably good, then many corrections can be fathomed by bound. Of course if a real incumbent is found by the search, then the search is equivalent to one without a pseudo-incumbent.

If the pseudo-incumbent is too low in cost, then because of the drastic pruning of the tree of corrections, no feasible correction will be found. This is in one sense a "wasted" search, but on the other hand, because the pseudo-incumbent prunes the tree so drastically, the search will require very little time. We have made a guess at the incumbent cost effectively for a number of problems.

Data Manipulation Methods

Our experience with the IPA system has shown that for certain integer programming problems, the effectiveness of the system can be increased if these problems are subjected to certain data manipulation methods. One of the most important problems which can be encountered in the use of IPA is that the determinant of the optimal LP basis may be extremely large. The reason this is a problem is because it is this determinant that determines the size of the abelian group which will be considered in the group optimization problem. For example, if this determinant is of the order of one hundred thousand, then the group network for any of the group optimization problems must include one hundred thousand nodes. This can result in extreme core storage requirements and hence can reduce the overall computational effectiveness of the system by requiring that at least part of the group optimization table be maintained on secondary storage. Such a situation is extremely undesirable for two reasons. First, in an extensive search through a large number of corrections the group table will be very heavily used by the backtracking algorithm. As a result, the system will require many disk accesses and hence, it will suffer a marked decrease in computational efficiency. This problem might not be so severe if it was possible to organize the group optimization table in such a way that only small blocks of it were required in core at a given time. In general, however, a path from a given node back to zero will pass through a number of nodes which are widely spaced in the group network. Therefore, unless we were willing to encode for each node the complete path from zero to that node for every backtracked correction the system would have to make some number of accesses to group nodes which are essentially randomly

selected. On the other hand, the prospect of encoding complete paths for each of the nodes in the network does not seem to be particularly appealing either because if the average path contains approximately five arcs, then we would be required to store approximately 1,000,000 entries for a group table which contained 100,000 nodes (five entries for he pathe and one entry for the cost). From the point of view of memory and computational time requirements, then, it is very adventageous to avoid dealing with problems in which the number of nodes in the group network is very large.

For these reasons we are particularly interested in restricting the size of the induced groups to a range of less than 10,000 in magnitude. Equivalently, we can say that we want to insure that the determinant of the optimal LP basis is less than 10,000. The difficulty of large determinants can be overcome by the application of some rather simple number theoretic procedures. Although these procedures cannot be guaranteed to reduce the size of the induced groups, computational experience to date has shown them to be efficacious in practically all cases. The central idea is that a given integer programming problem can be usefully transformed by dividing all the coefficients of an inequality by a rational divisor greater than one and then taking integer parts. Such a transformation of the problem is again a relaxation; that is, it has the property that all feasible solutions to the original IP problem are feasible in the transformed or relaxed problem.

Now consider the IP problem

$$\min \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad i = 1,\ldots,m$$

$$x_j \text{ non-negative integer.} \quad j = 1,\ldots,n$$

Let $\lambda_i > 1$ be a rational divisor for row i, i = 1,...,m. The following problem is a relaxation

$$\min \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} \left[\frac{a_{ij}}{\lambda_i}\right] x_j \leq \left[\frac{b_i}{\lambda_i}\right] \qquad i = 1,\ldots,m$$

$$x_j = \text{non-negative integer} \quad j = 1,\ldots,n$$

(See (3) for details.) Furthermore, an optimal solution to the relaxed problem which is feasible in the original problem is optimal in the original problem.

These results provide the basis for our data manipulation method which is directed at reducing the size of the group network. If the determinant of the optimal LP basis for a given IP problem is too large for the IPA system to deal with that problem effectively, then we can relax one or more rows of the original problem in hopes of reducing the determinant. We cannot guarantee that for any particular set of divisors the determinant will be decreased, however, because the determinant is not in general a monotonic function of the divisors. This is because we use integer parts in our division rather than real division. On the other hand, it is relatively easy to select some divisors which will produce a significant reduction in the size of the determinant. At present the IPA system permits the user to introduce a divisor for each row of the original problem. If a given row is relaxed, the new version of the row is introduced into the integer programming problem and the original row is retained as a side constraint. If the system finds a feasible solution to the relaxed problem, it automatically tests that solution on the original constraint. In the event that the solution is feasible for the original constraint, then the solution is feasible for the original problem. If, however, the solution fails this additional feasibility test, the search is continued using the relaxed problem.

Our experience using relaxation suggests that in many real world integer programming problems there exist one or some small number of constraints which cause the determinant of the LP basis to be quite large. For problems of this type we have found that the relaxation of these constraints has made the problem much simpler. On the other hand, it has been necessary to avoid making too severe relaxations on these problems because in doing so

one loses some of the basic structural aspects of the problem. A general appreciation of the use of relaxation can be obtained from the following example which is discussed in more detail in reference (3).

A brand allocation problem is formulated as an integer programming problem. The purpose of this problem is to help a large company equitably and profitably allocate television commercials to its different brands subject to a variety of logical constraints. The particular problem consisted of 86 constraints and 148 0-1 variables. Additional upper bound constraints on the variables were handled implicitly. Most of the constraints were logical constraints with non-zero coefficients equal to one. There were, however, four constraints with large coefficients of three or four place integers. No relaxation of these four constraints was done for the initial run, and the optimal LP basis had a determinant of the order of 10^{39}. These constraints were then relaxed until all of the non-zero coefficients were one digit. The determinant of the optimal basis of the relaxed problem was 380. Because of certain inaccuracies introduced by the relaxation, the solution to the relaxed problem was slightly infeasible in the original problem. Several other runs were made using different divisors for these rows and the determinant of the relaxed problem was in the range of 168 to 380. Five such runs*were required in order to find a solution which was considered to be satisfactory by the people who had formulated the problem.

As noted, we are presently using divisors to reduce the determinant of the LP basis to manageable size. This relaxation, however, is not automatic within the system. Before we can introduce such methods into IPA, we must make some of our ideas about the selection of divisors more precise. On the

---

*ranging from 29 to 78 seconds on the UNIVAC 1108 machine

other hand, we have found that ad hoc procedures for setting these divisors have been extremely useful in solving certain otherwise difficult integer programming problems. In conjunction with our attempt to make these procedures more formal we are also in the process of implementing a subroutine to perform relaxation of the original problem after the linear programming problem has been solved. The theoretical details of this procedure are presented in (3). Basically the idea is as follows. It is possible to solve the original linear programming problem and if the determinant is too large, to make a transformation of the optimal LP basis which is equivalent to that which would have been obtained had the problem been relaxed originally. This has a decided advantage of avoiding resolving the LP problem after the relaxation. We feel that the introduction of such a routine into the IPA system will be of significant value in the future.

There is another method which we have employed to control the size of the determinant of the basis which, strictly speaking, is not one of data manipulation. On the other hand, it is convenient to discuss this method at this point. The choice of a particular optimal LP basis when there is large scale LP degeneracy should not be left to the simplex method. Computational experience on a limited number of test problems has suggested the following procedure. Let the simplex method find an optimal solution and some set of optimal LP reduced costs. If the optimal basis determinant is too large, take the optimal basic activities which are at a positive level and pivot them into an identity matrix. The result is a basis with many basic slacks, and this basic solution is the same as the previously obtained optimal basic solution. The shadow prices relative to this basis may not be dual feasible, but the previously obtained optimal LP reduced

costs are non-negative and represent a consistent set of cost coefficients to use in subsequent analysis. (See Gorry and Shapiro (1)).

Here we will only present one example of the usefulness of this technique. A scheduling problem with 250 rows was found to have an optimal basis determinant of approximately 103,000. There was, however, large scale degeneracy, and the procedure suggested above led to an optimal basis corresponding to the same solution with determinant equal to eight. This reduction in the size of the induced group was more than sufficient to permit IPA to solve this problem.

Overall System Performance

To date, our experience with the IPA System has been very good. In dealing with certain problems, we have had to resort to the data manipulation methods or some problem reformulation. In a few other cases, we have found acceptable solutions which we did not test for optimality. With these reservations, we can say that IPA has solved all problems on which it has been used. Furthermore, these solutions have been obtained very efficiently relative to results reported in the literature on problems of similar size.

In Figure 4, we have presented some overall times on IPA performance for a few representative problems. These problems are typical of the kinds of "real world" problems with which we have been working.

Figure 4. EXAMPLES OF RESULTS WITH IPA

PROBLEM TYPE	No. of Constraints	No. of Variables	LP Time	Group Rep. Time	Group Soln. Time	Search Time	Total Time	Machine
MEDIA SELECTION	12	116	0.99	0.38	0.76	0.15	14.0	
	12	116	1.34	0.37	0.19	0.14	14.0	UNIVAC 1108
FIXED CHARGE	14	32	0.07	0.03	1.70	0.27	2.36	360/85
CAPITAL INVESTMENT	36	72	5.56	0.91	1.16	103.2	112.1	360/67
PERSONNEL SCHEDULING	57	132	6.86	-	-	2.85	33.0.	UNIVAC 1108
BRAND ALLOCATION	86	195	18.38	1.59	1.39	2.09	35.0	
	86	195	12.82	1.52	0.95	2.59	29.0	UNIVAC 1108
AIRLINE CREW SCHEDULING (2)	313	482	35.14	5.10	0.05	150.0	192.5	360/85

NOTES: 1. Times shown above are in seconds.
2. Feasible integer solution with a cost within 3.5% of the minimal LP cost found by search in time indicated. Optimality was not proven, run was terminated with a maximum time cutoff.

References

1. Gorry, G. Anthony and Shapiro, Jeremy F., "An Adaptive Group Theoretic Algorithm for Integer Programming Problems," Management Science, Vol. 17 (January 1971), pp. 285-306.

2. Gorry, G. A., Nemhauser, G. L., Northup, W. D. and Shapiro, J. F., "An Improved Branching Rule for the Group Theoretic Branch-and-Bound Integer Programming Algorithm," Mimeograph Report, October 1970. To appear in Operations Research.

3. Gorry, G. A., Shapiro, J. F. and Wolsey, L. A., "Relaxation Methods for Pure and Mixed Integer Programming Problems," Sloan School Working Paper 456-70, MIT, April 1970. To appear in Management Science.

4. Shapiro, J. F., "Dynamic Programming Algorithms for the Integer Programming Problem — I: The Integer Programming Problem Viewed as a Knapsack Type Problem," Operations Research, Vol. 16 (1968), pp. 91-102.